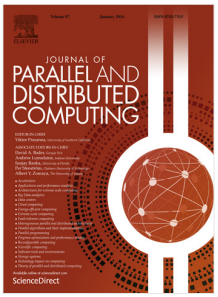
Accepted Manuscript



Online learning offloading framework for heterogeneous mobile edge computing system

Feifei Zhang, Jidong Ge, Chifong Wong, Chuanyi Li, Xingguo Chen, Bin Luo, Victor Chang

PII: S0743-7315(19)30149-2
DOI: https://doi.org/10.1016/j.jpdc.2019.02.003
Reference: YJPDC 4013

To appear in: *J. Parallel Distrib. Comput.*

Received date : 3 March 2018
Revised date : 6 December 2018
Accepted date : 19 February 2019

Please cite this article as: F. Zhang, J. Ge, C. Wong et al., Online learning offloading framework for heterogeneous mobile edge computing system, *Journal of Parallel and Distributed Computing* (2019), https://doi.org/10.1016/j.jpdc.2019.02.003

Online Learning Offloading Framework for Heterogeneous Mobile Edge Computing System

Feifei Zhang[a], Jidong Ge[a*], Chifong Wong[a], Chuanyi Li[a], Xingguo Chen[c], Bin Luo[a], and Victor Chang[b]

[a] *State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Nanjing 210093, China*

[b] *International Business School Suzhou and Research Institute of Big Data Analytics, Xi'an Jiaotong-Liverpool University, Suzhou, China*

[c] *School of Computer Science and Technology, School of Software, Nanjing University of Posts and Telecommunications, Nanjing, China*

Abstract

Cloud of Things (CoT) is a significant paradigm for bridging cloud resource and mobile terminals. Mobile edge computing (MEC) is a supporting architecture for CoT. The objectives of this paper are to describe and evaluate a method to handle the computation offloading problem during user mobility which minimizes the offloading failure rate in heterogeneous network. Furthermore, users' mobility and their choices for offloading lead to the everchanging condition of wireless network and opportunistic resource available. By modeling such dynamic mobile edge environment, quantizing the user cost, failure penalty and diversified QoS requirements, computation offloading problem is converted into an online decision-making problem in a stochastic process. We divide the decision-making into two phases: offloading planning phase and offloading running phase. In both phases the learning agent can continuously improve the control policy. We also conduct a failure recovery policy to tackle different types of failure and is included in the decision-making process. The numerical results show that the proposed online learning offloading method for mobile users can derive the optimal offloading scheme compared with the baseline algorithms.

Keywords: mobile edge computing, cloudlet, computation offloading, offloading failure, reinforcement learning.

1 Introduction

Mobile edge computing (MEC) is a network architecture enabling high bandwidth, agile mobile services, and low-latency which has attracted much attention in both academia and industry [1-3]. Recently, cloud-assisted Internet of Things (Cloud-of-Things or in short CoT) has emerged as a significant paradigm that enables intelligent and self-configuring (smart) IoT devices and sensors to be connected with the cloud through the Internet. MEC is closely related to this topic because it enables cloud computing service at the edge of cellular network, which provides timely and on-demand access for mobile applications during users' moving. The reason behind is that plenty of storage space and computing power are harvested dispersed at the edges of networks, which spread sufficient capabilities to mobile terminals' proximity and are especially beneficial for performing latency-critical and computation-intensive tasks. MEC architecture makes use of any available edge nodes and telecommunication networks for rapid deployment of services for customers. Among them, RAN (Radio Access Network) usually plays the role that authorizes third-parties, such as content providers.

Cloudlet is a kind of typical edge node in MEC architecture. We introduce two types of cloudlet and their features considered in this paper: ad hoc cloudlet and server-based cloudlet. "Data center in box" named server-based cloudlets [5] which widely deployed between mobile devices and centralized cloud over the Internet, is viewed as an edge server or cloud resource with powerful processors and fast transmission speed. Complementarily, cloudlet formed by a local mobile device

ad hoc network is called ad hoc cloudlet, which is especially useful when the Internet is inaccessible [5-6]. Based on the existing research of cloudlet, users have two choices to transfer their computation tasks for execution other than processing tasks in local mobile devices: server-based cloudlet execution and ad hoc cloudlet execution.

Computation offloading [4] recognizes the resource intensive part of a mobile application and dispatches the corresponding tasks to a cloud or edge servers through wireless links. The operation granularity can be task level, VM level or the whole application level. Correspondingly, the principle to guide such dispatching is called offloading control policy. From the perspective of a single user, offloading operation can be seen as an interaction between local mobile devices and mobile edge system. The resources ready for offloading can have many different organizational forms and types. In this paper, we consider the abstract platform —— cloudlet rather than concrete implemented products. In this paper, we consider the task level offloading.

These two types of cloudlets have their pros and cons: server-based cloudlet is more stable but not always available; ad hoc cloudlet is always accessible but its dynamicity can bring more uncertainty to task execution. Therefore, trade-off should be carefully made for offloading control policy to meet application demands and offer seamless mobile services such as opportunistic sensing and information integration, natural language /processing, machine learning, speech recognition, computer vision and graphics, augmented reality, reality and planning.

There are many challenges we are facing. For example, how to derive an optimal offloading policy in the dynamic and heterogeneous mobile edge system is still a hole need to be filled. “Heterogeneous” means that a mobile device usually has multiple wireless mediums, such as cellular communications, Wi-Fi and short-distance communication techniques. Different connection types perform varied in terms of energy consumption and processing speed. “Dynamic” means that the context——network connection and available resource change over time. This requires more complicated offloading control policy of switching wireless interfaces to provide users with high QoS and seamless service. In the context of MEC, QoS for mobile applications means low energy cost, rapid response and strong robustness. Seamless service means transparent switching between different contexts. As a concrete scenario, when it is infeasible to connect to the Internet, a mobile device user can also connect to a nearby server-based cloudlet to outsource the computation intensive mobile tasks. Alternatively, by setting up an ad hoc cloudlet, a group of mobile devices can still configure a computation offloading service. However, since users are in a fully dynamic environment where the context of a mobile device changes irregularly, timely decision making for computation offloading is needed to ensure that the decision making can always keep up. Especially, users’ movement can bring several problems like service switching, which leads to higher failure rate and is a weak link for task processing. Therefore, the following problems need to be addressed: 1) how to make timely and proper offloading decision? 2) how to reduce the failure rate of task execution?

Here we use a location-aware video recommendation application to demonstrate how our proposed problem fit the real scenario. In mobile video notification, the acceptance of pushed content is affected the contextual factors, including when and where the user receives the pushed contents, which type of device users choose to receive it. Time, location and device show great impact on user acceptance. If users open the function to receive annotations, user can check no more than 30 words of video’s abstract. Then users can click this message and jump to the mobile terminal’s app to watch this video or delete this video. Users select these two ways to explain

reception or rejection. Client App gives feedback to the news cooperation including user's behavior, context and basis context. The schematic diagram of this application is depicted in figure 1. The data comes from several news cooperation **groups.** In addition, to collect this information, news cooperation **decodes** the feedback from clients as JSON objects and store them as CSV files.

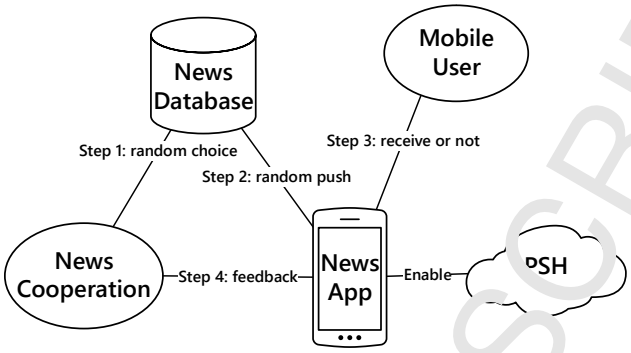


Figure 1. The procedure of mobile video push.

Based on the above analysis, we aim to obtain an optimal policy to minimize communication and computation costs as well as failure rate of the mobile applications. MEC environment includes server-based cloudlets and ad hoc cloudlets. At any given moment, there are several real-time applications to be executed. As an application could be divided into code sections (denoted as tasks), during the execution, the mobile user can decide to execute tasks locally on the mobile device, offload tasks to nearby server-based cloudlets or ad hoc cloudlets. A reinforcement learning based offloading framework is proposed in a heterogeneous MEC environment, which also handles the various failure cases in data transmission and task processing. Since the MEC environment is highly dynamic and unreliable, some uncertain factors cannot be properly included in one step learning policy, we present a multi-step learning and control policy which can revise errors and update by itself is presented. This framework divides the task offloading strategy into two phases: offloading planning phase and offloading running phase. In offloading planning phase, one-step reinforcement learning methods are adopted to derive an initial offloading strategy. During offloading planning phase, the learning agent keeps on learning to derive an optimal offloading strategy. Besides, the fault tolerant mechanism is involved in offloading running phase. The failure detection along with the failure recovery mechanism is jointly adopted to improve the successful rate of task execution. In offloading running phase, the learning agent continuously improves the control policy by adapting to actual offloading process. This phase further considers the waiting time of each tasks as well as fault tolerant mechanism. We evaluate the reinforcement learning based framework in both online and offline settings. For all we know, there is no existing work on online fine-grained code level offloading control policy in a dynamic MEC system with heterogeneous network resource, and explicitly considers user mobility and offloading failures. The inherent complexities have not been addressed which either optimize the heterogeneous MEC resource [7-8] and dynamicity of MEC environment separately [9] rather than taken these two aspects together. Most of all, only optimizing one factor cannot just satisfy the users' QoS requirements in a complex decision process. A way to navigate the three-target tradeoff between user monetary cost, energy consumption and failure penalty is needed.

The main contributions of this paper are listed as follows:

- We propose an online multi-step reinforcement learning based offloading framework for a single mobile user in an intermittently connected and unreliable MEC system to obtain an optimal offloading control policy. The policy includes cost expectation scheme to help determine offloading/local execution actions and execution sites based on the state of the single mobile user to achieve the minimum cost and failure rate. An estimation-based cost model is proposed to make sure several QoS goals are optimal.
- We consider the heterogeneity of the MEC environment and two typical MEC service providers: server-based cloudlet and ad hoc cloudlet. We provide a proper modeling for these two network architectures and consider them jointly for offloading control policy.
- The failure detection and recovery strategy is embedded in our proposed framework. We consider the backup-based failure recovery strategy in ad hoc cloudlet and use the checkpoint technique to do failure recovery in server-based cloudlet. Furthermore, a threshold-based fault tolerant mechanism is proposed to minimize failure happen rate during task execution.
- Two modified one-step algorithms — ϵ-greedy algorithm and soft-max algorithm are introduced in the multi-step learning framework to test performance.

We perform several examinations and validations of the proposed framework and prove that our solution is an optimal policy.

The rest of paper is organized as follows. Section 2 lists the related works from the perspective of MEC architecture, computation offloading and the heterogeneous and dynamicity of MEC environment respectively to state the novelty and significance of our work. Section 3 presents the system model. Section 4 introduces the problem formulation. Section 5 gives the online learning computation framework and associated algorithm. Section 6 shows the evaluation of the framework. Section 7 is the conclusion and future work.

2 Related work

2.2 Computation offloading in MEC.

Given the capability constraint of mobile terminals, offloading is a promise solution to expand the processing and storage capability of mobile terminals. A main theme of edge research is offloading policy on the user side, i.e., what/when/how to offload a user's workload from its device to the edge system or cloud [17]. Since MEC is a newly arise concept, few works focus on the computation offloading problem in MEC environment [7], [18-25]. Most of them are focused on the energy minimization, coordination among multiple users and combining computation offloading with other technique. The methods proposed for computation offloading are divided into two kinds according to their purpose: frameworks and algorithms. Gao et al. [25] proposed an online data offloading algorithm (NDO). Wifi connection is seemed as an opportunistic resource in their work. Wang et al. [26] think that one promising approach to deal with the intermittent connectivity and wireless coverage is to offload computation to nearby mobile devices. Chen et al. [27] designs a peer-to-peer mobile cloudlet communication model based on short-range radio communication which interconnects nearby mobile devices. Jin et al. [6] are interested in ad hoc cloudlet networking and propose a dynamic cloudlet self-networking framework to configure computation offloading. Taking node mobility into account, dynamic cloudlet behavior is investigated, they design an optimized allocation algorithm called SA-UM to reduce the complexity of resolution space on component allocation algorithm.

2.1 Mobile edge computing: architecture.

Mobile edge computing proposed in 2014 [10] is known as a distributed edge server system which

allows mobile terminals (e.g., tablets and smartphones) to migrate their task processing and data storage to edge servers. To unlock the potential of edge computing, cloudlet [5], [11], originally a popular concept in mobile cloud computing, has been considered in MEC system for its proximity to mobile users. There is a creeping heat in the field of MEC. Recently, the definition of edge devices becomes wider, any devices that have computing resources along the path between data sources and cloud data centers can be encompassed in MEC architecture [12]. MEC has the advantages of saving energy, achieving a lower latency, enhancing privacy, supporting context-aware computing for COT applications [12]. Fog computing [13] is a concept related to the same paradigm. The domains of MEC and fog computing are overlapping and the terminologies are frequently used interchangeably and provide architecture for C-IoT implement.

There is a growing trend that seeing the hybrid network architecture as a whole for computation offloading or business data processing [14-16]. Ku et al. [14] introduce the system design of the radio access network (RAN) with the fog computing paradigm. Based on the CPU loads and traffic concerns, Chiang et al. [15] discuss the pros and cons of hybrid network in fog-cloud environment. They suggest that short latency, enough network bandwidth, and geographic locality, scalability and modularity are important requirements for fog architecture. Fog may form a hierarchical architecture for C-IoT.

2.3 Dynamic and heterogeneous network condition

Most of the aforementioned MEC offloading schemes didn't take the heterogeneous dynamic MEC environment into account. Unlike the stable integral data centers, MEC environment is essentially hybrid and opportunistic. However, the study about computation offloading for a single user in heterogeneous MEC is still immature. There are many existing works using reinforcement learning method to deal with the changeful mobile computing system. As is mentioned in [28], intermittent issues may due to the user mobility, device connection policy and dynamic changing network condition. Terefe et al. [29] use a discrete time Markov chain (DTMC) to model fading wireless mobile channels. To solve limited battery problem, Chen et al. [27] present a semi-Markov decision process (SMDP)-based optimization framework aimed for modulation scheme transmission bit rates and various DVFS levels, in order to minimize both the average latency and the energy drawn from the battery. Sun et al. [30] design a latency-aware workload offloading (LEAD) strategy to dispatch mobile application tasks into proper cloudlets. In [30], a stochastic model is developed to study the dynamic offloading in the context of MCC and the problem of intermittently available access links is well addressed. Aiming for intermittently connected environment, Zhang et al. [28] also provide a strategy to minimize user computation cost and communication based on MDP model. They also use a threshold strategy to reduce the complexity of MDP model. Liu et al. [32] present a reinforcement learning-based resource management algorithm to decompose the learning approach to two parts: (offline) value iteration and (online) learning approach. Wang et al. [26] think that devices' mobility has regulated contact patterns and can be used to solve mobility-assisted opportunistic computation offloading problem. Most of the aforementioned reinforcement learning based offloading scheme use MDP model to fit the stochastic process embedded in mobile computing system, but they ignore the heterogeneous of environment and the problem about how optimally utilize the heterogeneous resources. Hu et al. [33] study the network selection problem in operator-initiate offloading in ultra-dense wireless networks. However, these reinforcement learning-based approaches are only adopted in the offloading planning phase, which means that the learning agent cannot adaptively improve the existing strategy. This will make the control policy

less likely to adopt to the outer environment.

3 System models

Under our consideration, the MEC system model investigated in this paper is composed of three elements: radio access network attached with server-based cloudlets, ad hoc cloudlets and the user mobile device. Considering a mobile user with a limited task queue, a processor, a data transceiver, and is moving in a semi-determined pattern [34], [37], [38]. We first explain the following terminologies which are used throughout of the paper. Furthermore, main mathematical notations used throughout the paper are introduced in table 1.

3.1 Terminologies

In this section, we introduce the involved major terminologies in MEC architecture.

Mobile devices. The mobile devices can be small hand-held devices or bigger portable devices.

Access points. Generalized access point can be base station in cellular network or Wi-Fi access point. Some of them are attached with cloudlet system.

Short-distance communication technique. There are many existing short-distance communication technique, e.g., Bluetooth and Wi-Fi, etc. the transfer distance of Bluetooth is 10 meters, IrDA is 1 meter, while Wi-Fi has advantage in wave coverage, which live up to 100 meters. Zigbee is 10-1000 meters. In reality, there could be more than one communication techniques adopted by mobile users. We **do not** assume the exact communication technique and consider a general case in our work.

Cloudlet. The key idea is to move part of or the whole data and the computation tasks from mobile terminals to the cloud in the same local area network (LAN) transparently and seamlessly, to solve the limited resource problem of mobile terminals. Cloudlets are usually deployed in one-hop proximity from mobile users. It can be classified into two types by its way of organization, named server-based cloudlet and ad hoc cloudlet.

Table 1 Main Mathematical Notations in this paper.

Notations	Definition
$S=(Z,Q,V,N)$	Composite state of a mobile user, which includes queue length Q, the number of users Z, the number of service nodes available V and the number of connections to an access point N.
$A,\ A(S)$	Action space includes local execution mode, server-based cloudlet execution mode and ad hoc cloudlet execution mode. for each state S, $A(S)$ is the available action set.
$C(S,A)$	The immediate cost of (S,A).
λ_q	Job arrival rate to the queue.
λ_c	Node contact rate to the mobile user.
λ_n	Node connection rate to the access point.
δ_i	The congestion level of access point i which user connected with.
Φ	The coverage area of an access point.
$t=(wl,ip,op,ar,ls)$	Real-time task t with certain workload wl, input and output data size ip and op, arrival time ar and lifespan ls.
γ	In range $\{0,1\}$, $\gamma=1$ means the task finally returns correct result, otherwise, the task execution is failed in the end.
β	The failure threshold set for the fault tolerant policy.

e_{up}, e_{down}, e_{ad}, e_{cpu}	Unit energy consumption of uplink data transmission e_{up} and downlink data transmission e_{down}, short-distance communication e_{ad} and local execution e_{cpu}.
bw_{up}, bw_{down}	The total uplink bandwidths bw_{up} and downlink bandwidths bw_{down} of an access point.
s_{ad}	The processing speeds of ad hoc cloudlet nodes.
κ	The maximum execution counts of tasks.
c_{pen}, c_{dt}	The failure penalty for one failure execution happens c_{pen} and the penalty for not sending back correct result c_{dt}.
p_s	The probability that a node in the ad hoc cloudlet is willing to provide service for the user.
s_{cpu}, s_{ser}	The processing speed of local execution s_{cpu} and the processing speed of the server-based cloudlet s_{ser}.
c_{ser}, c_{ad}	The unit monetary cost of server-based cloudlet c_{ser} and ad hoc cloudlet c_{ad}.
c_{ran}	The unit monetary cost of data transmission through RAN.
f_{up}, f_{down}, f_{sd}, f_{ad}, f_{ser}, f_{cpu}	The failure rate of uplink data transmission, downlink data transmission f_{down}, short-distance communication technique f_{sd}, node execution in ad hoc cloudlet f_{ad}, server-based cloudlet execution f_{ser} and local execution f_{cpu}.

3.2 Mobile edge system

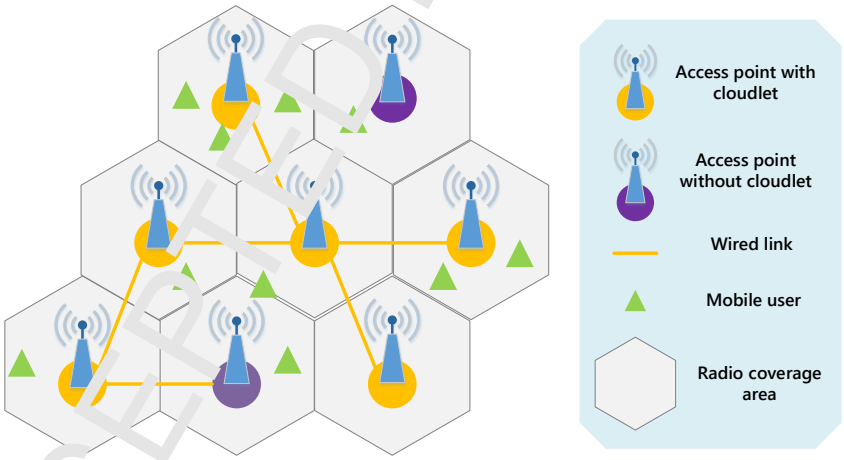


Figure 2. The aerial view of RAN layout.

From a user's point, mobile edge system is a main interaction object for computation offloading. The mobile users connect with server-based cloudlets through high bandwidth and low latency WLAN (or cellular links), and they can communicate with each other through short-distance communication links within an ad hoc cloudlet.

Since the focus of this paper is at the edge of network, RAN is an important part of telecommunication system. We define a marked undirected graph to describe the RAN structure and the deployment of server-based cloudlets. As shown in figure 2, in RAN layout, access points can

be divided into two types: 1) access point without server-based cloudlet which is denoted as a yellow circle, which means this access point lack of task processing capability but can be used as a router excepts that it is an isolated node; 2) access point with a server-based cloudlet, which is denoted by a purple circle, which capable of task processing. The edge between access points means a wired connection and the data transmission between is allowed.

For the deployment strategy of access points contained in RAN, we assume that the 2-D area is seamlessly covered by all the Z fixed homogeneous access points, and the access points' coverage regions do not have overlap with each other. The ids of zones are denoted by a set $\mathbb{Z} = \{1, \dots, Z\}$. The area covered by the signal of access point is called a "zone". Hence, the zone and the access point share a one-to-one mapping, the access point and the zone where the access point located in has the same id. The total available uplink and downlink bandwidth of an access point are denoted as BW_{up} and BW_{down}, respectively. The cost for unit data transmission is c_{ran}. The congestion level of access point in zone i is $\delta_i = 1/N_i$, where N_i is the number of users (a mobile user is denoted as a triangle in the figure 2 and 3) connected to access point in zone i (regional load) and $i \in [1, Z]$. For simplicity, the computation capability of server-based cloudlets is assumed as the same. When a task is about to be offloaded to the server-based cloudlet, it can only enter the cloudlet connected to the access point of the user dwelled zone. Usually, the coverage range of access points is much larger than short-distance communication technique.

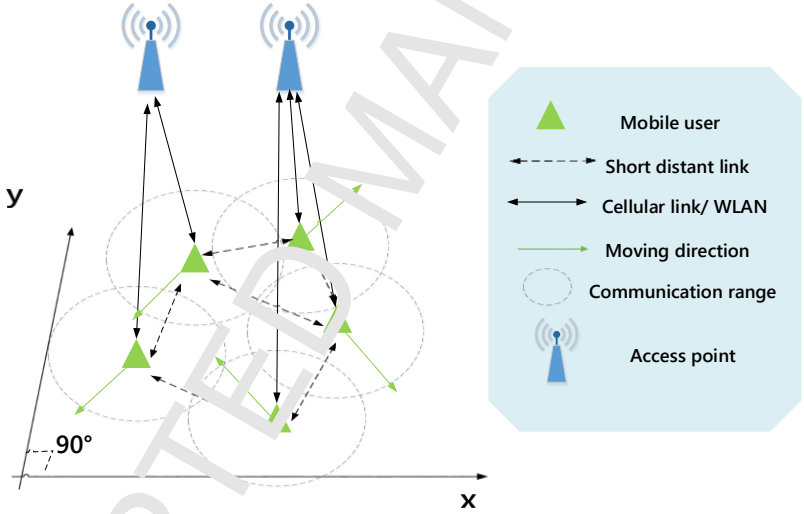


Figure 3. The network architecture of an ad hoc cloudlet.

Figure 3 gives the typical scenario of an ad hoc cloudlet, which is composed of a set of mobile device nodes. Each node can join or leave the ad hoc cloudlet at any time and can communicate with each other within the same ad hoc cloudlet through peer-to-peer short-distance communication technique. Furthermore, in the case that the service node roams to a different cell that is covered by Wi-Fi, e.g., the mobile user goes back home, the intermediate result can be uploaded to the immediately connected access point via Wi-Fi. In an ad hoc cloudlet, not all nodes are available for providing service. Furthermore, when a mobile user asks for service, only the nodes within its contact range can provide service. As user joining ad hoc cloudlet is not mandatory, the mobile devices within the user's contact range has a probability p_s to provide service to the user. Compared with server-based cloudlet, ad hoc cloudlet is highly dynamic, its device nodes can join or leave at any time.

Like many other distributed systems, ad hoc cloudlet adopts the fault tolerant strategy to guarantee the reliability of task processing. In this paper, we consider the replication-based fault tolerant strategy, which means that there are multiple copies in an ad hoc cloudlet. Once an execution fails, other backup copies will be activated and run. Supposed that there are n copies in the ad hoc cloudlet. the single point failure rate is f_{ad}, the transmission failure rate of short-distance transmission is f_{sd}. Then the execution failure rate is ${f_{ad}}^{n}$ and data transmission failure rate is ${f_{sd}}^{n}$. Since the multi-hop offloading is usually unreliable, we only consider the one-hop computation offloading. User can offload the current task to one service device in the ad hoc cloudlets as long as they are within the contact range of each other. The offloading strategy fully considers the dynamicity of ad hoc cloudlet, so each time mobile user offloads the task to the nearest service node for execution.

3.3 User mobility model

Several existing works reveal that the movement of a user usually follows a semi-deterministic pattern rather than a random way [35]. Yuan et al. [35] mentioned that the user movement between different social community can be described by transition probabilities in each time period. As an observation, an access point or base station usually covers a specific social place like community, such as library, dining hall, etc. At any zone, user could pick to stay for a while or move to another zone according to its preferred probability.

3.4 Mobile device and tasks

The user mobile device contains a processor with processing speed s_{cpu}, components such as cache and memory, a single-server FCFS queue Q to store arrival applications pending for execution, a short-distance communication interface and a wireless interface. We assume that mobile applications' arrival follows a Poisson process with parameter λ_q.

The unit energy consumption of task processing is e_{cpu} and the unit energy consumption of uplink and downlink data transmission through RAN are e_{up} and e_{down}, respectively. The unit energy consumption of short-distance communication is e_{ad}.

Real-time applications widely exist in mobile computing system. We model the applications being offloaded as independent workloads and can be partitioned into tasks. Thus, let t denote the task generated by the application, $t = (wl, ip, op, ls, ar)$, where wl is the number of instructions, ip and op are the input and output data size, respectively, ls is the lifespan and ar is the arrival time of task t.

3.5 Decision perioa

Since we consider an online control policy for computation offloading. We divide the control process into several equally-spaced time intervals named "decision period". At the beginning of each decision period, the task dispatching place is decided. It is assumed that the task can be finished in one decision period.

4 Problem formulation for optimization model

In this section, we formulate the dynamic offloading as an online learning problem and come up with a reinforcement learning framework to reduce the cost. After defining state space, action space and reward model, the stochastic control problem can be casted into a multi-armed bandit problem. Our hybrid offloading algorithm consists of two parts: offloading planning phase and offloading running phase. Here we adopt one-step methods in offloading planning phase: the modified ϵ-greedy algorithm or soft-max algorithm to derive an initial offloading strategy. Different from previous one-step methods, the learning agent keeps learning and improves the control policy

continuously in offloading running process. To reduce the searching space, the initial offloading strategy does not consider task waiting time and fault tolerant mechanism.

4.1 Components

The learning process are divided into two phases: offloading planning phase and offloading running phases. As depicted in figure 4, there are six components running on the user mobile device: offloading planner, offloading executer, task manager, learning agent, service aware and context monitor. Their functions and interactions are introduced as follows. All the six components run in user's mobile device.

Offloading planner. Offloading planner self-learns an offloading policy with limited knowledge about mobile edge system. It can fetch the system information, then provide the offloading executor with the initial policy and pass parameters to the agent for it can keep on learning in the offloading running phase.

Offloading executor. Offloading executor decides execution site for arrival task in actual scene. In each decision period, offloading executor fetches the environmental information from context monitor, service aware and task manager, then detect the system state. The system state is then passed to the agent for ongoing learning. Offloading executor use the current control policy to decide the task execution site.

Agent. An intelligent agent that performs reinforcement learning strategy plays an ontological role in both phases. The learned control policy is stored in the database. In each learning period, it updates the control policy until one of the stop conditions met.

Task manager. Task manager supervises the mobile device queue and the arrival tasks, makes sure that tasks are executed in order and abolishes the failure or timeout tasks. According to the control policy, it allocates the execution site for one task in each decision period and sends back the queue state to the offloading executor.

Service aware module. Service aware module discovers the available network service (server-based cloudlets and ad hoc cloudlets) for execution.

Context monitor. It monitors the task execution conditions and fetches the context information for offloading executor, so it can make adjustment timely such as failure recovery.

Database. It stores the current control policy, parameter settings and iteration information, so that the learning in both planning phase and running phase can convergence properly.

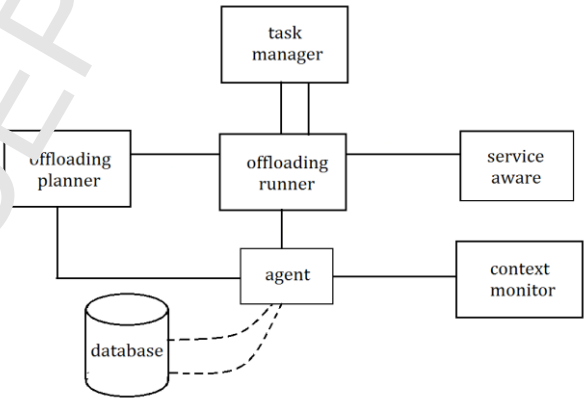


Figure 4. The online learning framework components.

4.2 State space and action space

The state space of the mobile user in the MEC system is defined as follows:

$$\mathbb{S} = \{S = (Z, V, Q, N) | Z \in \mathbb{Z}, V \in \mathbb{V}, Q \in \mathbb{Q}, N \in \mathbb{N}\}. \quad (1)$$

which is a composite state of a mobile user, including dwell zone $z \in \mathbb{Z} = \{1,2,3, \ldots, Z\}$, $v \in \mathbb{V} = \{0,1,2, \ldots, V\}$ denotes the number of contact nodes, queue length $q \in \mathbb{Q} = \{0\} \cup \{1,2, \ldots, Q\}$ and the number of nodes connected to the access point $n \in \mathbb{N} = \{0\} \cup \{1,2, \ldots, N\}$ in the current zone, where Z, V, Q and N are the maximum value of $\mathbb{Z}$, $\mathbb{V}$, $\mathbb{Q}$ and $\mathbb{N}$, respectively.

In each decision period, the number of waiting tasks can increase until the queue is full (i.e., $q = |\mathbb{Q}|$).

We assume that the geographical distributions of mobile nodes in ad hoc cloudlets follows an independent homogeneous Poisson point process (HPPP) [35-36]. The number of mobile nodes is distributed as an HPPP. We assume that at the beginning of each decision period, the probability of the size of contact node set is calculated as

$$\mathrm{P}(V = v) = e^{-\pi r^2 \lambda_c} \cdot (\pi r^2 \lambda_c)^v \; v!. \quad (2)$$

Where λ_c is the node distribution density of the ad hoc cloudlets and r is the communication range of user's mobile device.

The number of connections to the access point can influence the actual bandwidth allocated to the service users. Here we calculate the average bandwidth for the connected users. The more connections, the fewer bandwidth the user can use. The upper bound of bandwidth for a user is bw_{max}. Supposed that the coverage of access points does not overlap of each other, we can use a 2-dimentional spatial point process to simulate the number of connections to an access point in each decision period:

$$P(N = n) = e^{-\Phi} \cdot (\Phi \cdot \lambda_n)^n / n!. \quad (3)$$

where Φ is the area of the access point's coverage range.

The action space of MEC system is derived as

$$\mathbb{A} = \{A = 0, A = 1, A = 2\}, \quad (4)$$

which denotes that a mobile user can make a decision to execute a task locally on a mobile device (i.e., $A = 0$), offload to the accessible server-based cloudlet (i.e., $A = 1$) or offload to the available ad hoc cloudlet (i.e. $A = 2$). Note that in some cases, not all the action can be chosen. For example, user enters a zone without server-based cloudlet, then he or she only has choices to execute task locally or offload it to the ad hoc cloudlet.

4.3 Failure recovery-based immediate cost calculation

The possible failure scenarios in the MEC system could be: 1) failure happens during task processing procedure; 2) the user has moved out of the coverage range of the current access point or server node so the output of this task cannot be returned.

To achieve the smart and elastic cost model $C(S, A)$, it is necessary to involve more charging factors. The proposed reward model is mainly composed of three components: user immediate monetary cost, immediate energy consumption and immediate failure penalty. As the mobile users may execute their computation tasks on one of the possible execution sites: locally ($A = 0$), remotely on server-based cloudlet ($A = 1$) or on the other mobile device in ad hoc cloudlet ($A = 2$). Please note that in some decision periods, not all the network resources are available. As for failure recovery, it is assumed that the failure happens are independent of each other. Once a failure happens, the task processing enters a failure recovery phase. However, failures could also happen in failure recovery phase. In this case, the failure recovery procedure should be repeated until the correct execution result returns to user or ends automatically due to timeout. Here we adopt checkpointing

technique to do the failure recovery which is widely adopted in mobile computing system to provide reliable service.

At the beginning of each decision time period, user makes decision about the execution position of the current task. It is assumed that the hybrid network is composed of RAN and WLAN/MANET. Those two networks can exchange data with each other. During task execution, the rules below must be obeyed.

1) One task can only be executed in one execution site during the lifecycle, including failure recovery period.
2) The upper bound of execution counts κ is the same for all the arrival tasks. The next failure recovery process can be triggered if and only if the current execution is failed. Otherwise, the execution result is returned and the execution life is over.

Based on the above analysis, the delivery of a computation task incurs a certain number of immediate costs:

1) User energy consumption, C_e: this is an elastic energy cost concerning task processing energy consumption and data transmission energy consumption.
2) Resource requisition cost, C_r: according to the contract of resource requisition in different networks, a certain amount of monetary cost should be paid in terms of network resource usage.
3) Failure penalty, C_f: failure could happen during the task execution and the data transmission. Once the failure happen, penalty should be paid according to the failure type. it uses two penalty factors to guide the agent to do the wisdom decision that minimize the failure happens. c_{pen} is the penalty for a failure execution, c_{dt} is the penalty for not returning the correct result.

As all the three costs have to be minimized simultaneously, the reward is formulated as a weighted sum of those costs:

$$C(S,A) = \omega_e \cdot C_e + \omega_r \cdot C_r + \omega_f \cdot C_f. \tag{5}$$

ω_e, ω_r and ω_f are the weighted factors which reflect the ratio of corresponding cost in the reward model, where $\omega_e + \omega_r + \omega_f = 1$.

Offloading control policy has to decide the execution site for the current task at the beginning of decision period. An important assumption we shall make is that all the tasks can finish its execution within a decision time period and return a successful or failure result if possible. User device will be informed when the current task is at the end of its life. For all the tasks, they have the same upper bounds of execution time κ. However, they have to finish executions within their predefined lifespans. The task execution can be repeated until 1) the upper bound of execution time is met; 2) the task execution returns the correct result; or 3) the lifecycle of a task is over.

Supposed that κ_e', κ_i' and κ_{op}' are the execution recovery counts, input data transmission recovery counts and output data transmission recovery counts, respectively.

In the following, we introduce the calculation of the immediate cost $C(S,A)$ by considering the following three cases.

Local execution ($A = 0$). If the local execution belongs to the available action in state S, then the only energy consumption for local execution is task execution and there is no need to transmit data. Therefore, the energy consumption of one execution is $e_{cpu} \cdot wl/s_{cpu}$, where e_{cpu} is the unit energy consumption of user mobile device, wl is the workload of task t and s_{cpu} is the processing speed of mobile device. Since we adopt failure recovery mechanism, a task can be executed multiple times until it returns the correct result or one of the finish conditions is met. Supposed that the failure execution count is κ_e' ($\kappa_e' = \kappa' - \gamma$) where $\gamma \in \{0,1\}$ is a factor

indicating whether the task finally returns the correct result ($\gamma = 1$ means returning the correct result).

The immediate cost for local execution can be derived as:

$$C(S,0) = \omega_e \cdot \kappa' \cdot e_{cpu} \cdot wl/s_{cpu} + \omega_f \cdot (\kappa'_e \cdot c_{pen} + \gamma \cdot c_{dt}), \quad (6)$$

where c_{pen} is the penalty for a failure execution, c_{dt} is the penalty for not returning the correct result to the user during the task's lifecycle. $C_e = \kappa' \cdot e_{cpu} \cdot wl/s_{cpu}$ is the immediate energy consumption; $C_f = \omega_f \cdot (\kappa'_f \cdot c_{pen} + \gamma \cdot c_{dt})$ is the immediate failure penalty.

Server-based cloudlet execution ($A = 1$). If server-based cloudlet execution is available in state S, then its calculation can be derived as follows. First, we consider successful execution. The energy consumption includes uplink data transmission energy consumption $e_{up} \cdot ip/BW_{up} \cdot \delta$ and downlink data transmission energy consumption $e_{down} \cdot op/BW_{down} \cdot \delta$. The scheduling cost is $c_{ran} \cdot ip/BW_{up} \cdot \delta + c_{ser} \cdot wl/s_{cloudlet} + c_{ran} \cdot op/BW_{down} \cdot \delta$, where BW_{up} is the total uplink bandwidth of an access point, BW_{down} is the total downlink bandwidth of an access point, δ is the congestion level of the zone user dwell in. c_{ser} and c_{ran} are the unit cost of task processing and data transmission through RAN, respectively.

The failure execution can be divided into three types according to the phase where failure happens.

1) Input data transmission failure. If task is offloaded to the server-based cloudlet, input data transmission failure could happen during task processing. If such failure happens, the following computation and data transmission of this task will be abolished and a failure message is returned. As the features of returned message is orthogonal to our research, here we assume that the failure message adds no overhead, referring to the communication mode chosen, the energy consumption for input data failure is $e_{up} \cdot ip/BW_{up} \cdot \delta$ for wireless transmission.
2) Execution failure. Only when the task finishes its execution the result can be returned, therefore, the energy consumption for failure execution is $e_{up} \cdot ip/BW_{up} \cdot \delta$.
3) Output data transmission failure. If the task is offloaded to the cloudlet or MANET, output data transmission failure could happen. Since we adopt a coarse-grained failure recovery mechanism, in this case, the energy consumption is the same as the successful execution.

Here we assume that the code and data offloaded are correct, the factors led to failure happen only come from the environment. The failure recovery phase needs no more monetary cost from user but only requires more energy consumption.

Based on the analysis above, supposed that input data transmission failure happens κ'_{ip} times, execution failure happens κ'_e times and output data transmission failure happens κ'_{op} times. Then the total recovery counts $\kappa'_f = \kappa'_{ip} + \kappa'_e + \kappa'_{op}$. If $\kappa'_f + \gamma < 0$, then the task is immediately abandoned and its immediate cost is set to zero.

Therefore,

$$C(S,1) = \omega_e \cdot C_e + \omega_r \cdot \kappa' \cdot C_r + \omega_f \cdot \left(\left(\kappa'_{ip} + \kappa'_e + \kappa'_{op}\right) \cdot c_{pen} + \gamma \cdot c_{dt}\right), \quad (7)$$

Where

$$C_e = \left(\kappa'_{ip} + \kappa'_e + \kappa'_{op} + \gamma\right) \cdot e_{ip} \cdot \frac{ip}{BW_{up}} \cdot \delta + \left(\kappa'_{op} + \gamma\right) \cdot e_{op} \cdot \frac{op}{BW_{down}} \cdot \delta. \quad (8)$$

$$C_r = c_{ran} \cdot ip/BW_{up} \cdot \delta + c_{ser} \cdot wl/s_{ser} + c_{ran} \cdot op/BW_{down} \cdot \delta. \quad (9)$$

$\gamma \in \{0,1\}$ is a factor indicating whether the task returns the correct result, 1 for successfully return and 0 for returning error or no result returns.

Ad hoc cloudlet execution ($A = 2$). If ad hoc cloudlet execution is available in state S, then its

calculation can be derived as follows. Firstly, we consider one successful execution. There are two ways for data transmission, one is through radio access network using wireless connection and the other is through ad hoc network using short-distance communication technique. For wireless connection, the calculation for energy consumption is the same as server-based cloudlet execution. Since the receiver node in ad hoc cloudlet receives data and code through RAN network, data transmission through RAN doubles the data transmission time. The scheduling cost is $2 \cdot c_{ran} \cdot ip/BW_{up} \cdot \delta + c_{ad} \cdot wl/s + 2 \cdot c_{ran} \cdot op/BW_{down} \cdot \delta$, where BW_{up} is the total uplink bandwidth of an access point, BW_{down} is the total downlink bandwidth of an access point, δ is the congestion level of the zone user dwell in. c_{ad} is the unit monetary cost of task processing in ad hoc cloudlet, s is the processing speed in ad hoc cloudlet node and $s \in s_{ad}$. The energy consumption for one successful execution is $2 \cdot e_{up} \cdot ip/BW_{up} \cdot \delta + 2 \cdot e_{down} \cdot op/BW_{down} \cdot \delta$. For short-distance communication data transmission, energy consumption is $e_{ad} \cdot ip/r_{ad}$ for input data transmission and $e_{sd} \cdot op/r_{ad}$ for output data transmission, where r_{ad} is the data transmission rate and $r_{ad} \in \Omega$, e_{sd} is the unit energy consumption of short-distance communication. The scheduling cost is $C_r = c_{ad} \cdot wl/s$, where c_{ad} is the unit monetary cost for task execution.

For failure cases, similar to that of server-based cloudlet execution, can be divided into three types as follows. Here we compute the energy consumption for one failure execution.

1) Input data transmission failure. the energy consumption for input data failure is $e_{up} \cdot ip/BW_{up} \cdot \delta$ for wireless transmission, $e_{ad} \cdot ip/r_{ad}$ for short-distance communication.
2) Execution failure. Only when the task finishes the execution the result can be returned, since user need not to receive result data, the energy consumption is the same as input data transmission failure.
3) Output data transmission failure. The failure happens when user receives the output data, here we consider from coarse granularity level and assume its energy consumption is the same as the successful execution.

Based on the analysis above, supposed that input data transmission failure happens $\kappa_{ip}{}'$ times, execution failure happens $\kappa_e{}'$ times and output data transmission failure happens $\kappa_{op}{}'$ times.

$$C(S,2) = \omega_e \cdot C_e + \omega_r \cdot \kappa' \cdot C_r + \omega_f \cdot \left(\left(\kappa'_{ip} + \kappa'_e + \kappa'_{op}\right) \cdot c_{pen} + \gamma \cdot c_{dt}\right). \quad (10)$$

For RAN data transmission, the energy consumption is calculated as

$$C_e = \left(\kappa'_{ip} + \kappa'_e + \kappa'_{op} + \gamma\right) \cdot e_{ip} \cdot \frac{ip}{bw_{up}} \cdot \delta + \left(\kappa'_{op} + \gamma\right) \cdot e_{op} \cdot \frac{op}{bw_{down}} \cdot \delta. \quad (11)$$

For short-distance communication, the energy consumption is

$$C_e = \left(\kappa'_{ip} + \kappa'_e + \kappa'_{op} + \gamma\right) \cdot e_{ad} \cdot \frac{ip}{r_{ad}} + \left(\kappa'_{op} + \gamma\right) \cdot e_{ad} \cdot \frac{op}{r_{ad}}. \quad (12)$$

4.4 The expected immediate cost

As the decision making is based on the immediate cost, it should be derived before decision making. However, some of the calculated values can only be derived after task execution. To solve this contradiction, we use the expected immediate cost instead when making decisions.

Because the appearance of failure is an uncertain event, we use an expectation to approximate the reality scenario. Similarly, we discuss the expectation calculation separately for different task execution sites.

Local task execution. Given that the local execution failure rate is f_{cpu}, the expected value of C_e is calculated as follows:

$$\mathbb{E}(C_e) = e_{cpu} \cdot wl/s_{cpu} \cdot \left(1 - f_{cpu}\right) \cdot \sum_{n=0}^{\kappa'-1} {f_{cpu}}^n \cdot n + e_{cpu} \cdot wl/s_{cpu} \cdot {f_{cpu}}^{\kappa'} \cdot \kappa'. \quad (13)$$

where κ' is the maximum execution times of local execution. $\left(1 - f_{cpu}\right) \cdot \sum_{n=0}^{\kappa'-1} {f_{cpu}}^n \cdot n$ on the left side is the expectation of execution counts under the condition that finally the correct result can be returned to mobile user. As we view failure recovery at a coarse level, the energy consumption of failure execution and successful execution are viewed as the same. The term on the right side is the expected energy consumption when all the executing attempts are failed, where ${f_{cpu}}^{\kappa'}$ is probability that all the κ' attempts are failed.

For the immediate scheduling cost C_r, no matter how many times the task executes, it only calculates once. However, when $\kappa' = 0$, $C_r = 0$, the expected failure penalty C_f can be derived as

$$\mathbb{E}(C_f) = c_{pen} \cdot \left(1 - f_{cpu}\right) \cdot \sum_{n=0}^{\kappa'-1} {f_{cpu}}^n \cdot n + c_{pen} \cdot {f_{cpu}}^{\kappa'} \cdot \kappa'. \quad (14)$$

Similarly, $\left(1 - f_{cpu}\right) \cdot \sum_{n=0}^{\kappa'-1} {f_{cpu}}^n \cdot n$ on the left side is the expectation of execution counts under the condition that finally the correct result can be returned to mobile user. ${f_{cpu}}^{\kappa'}$ is the probability that all the κ' attempts fail.

The calculation of ad hoc cloudlet execution and server-based cloudlet execution can be put together, because they both have more than one failure type with various failure recovery time lengths. As there are many possible types of failure with unknown occurrence numbers, the enumeration of all the possible error situations is a NP problem. For simplicity, we divide the task execution into two cases, failure execution includes all the types of failure and successful execution.

According to the calculation rules, only the energy consumption e_s and the expected execution time τ_e of the normal execution case, expectation of failure happen rate $\mathbb{E}(p_f)$, failure recovery time $\mathbb{E}(\tau_f)$ and failure recovery energy consumption $\mathbb{E}(e_f)$ are needed. And the expected maximum failure recovery counts can be derived as

$$\kappa_f' = \lfloor (ds - wait - \gamma \cdot \tau_s)/\mathbb{E}(\tau_f) \rfloor. \quad (15)$$

$\lfloor \ \rfloor$ means the lower integral bound, $wait$ is the waiting time of the current task, $\gamma \in \{0,1\}$ is a factor indicating whether the correct result is returned ($\gamma = 1$ returns the correct result and $\gamma = 0$ returns error or no result). If $\kappa_f' + \gamma < 0$, then the task is immediately abandoned and its immediate cost is set to zero.

Hence, the expectation of energy consumption is

$$\mathbb{E}(C_e) = \gamma \cdot (e_s \cdot \tau_s \cdot (1 - p_f) + \mathbb{E}(e_f) \cdot \mathbb{E}(\tau_f) \cdot \sum_{n=0}^{\kappa_f'} {p_f}^n \cdot n) + (1 - \gamma) \cdot \mathbb{E}(e_f) \cdot \mathbb{E}(\tau_f) \cdot {p_f}^{\kappa_f'} \cdot \kappa_f'. \quad (16)$$

$e_s \cdot \tau_s \cdot (1 - p_f)$ is the expected energy consumption of successful execution, $\mathbb{E}(e_f) \cdot \mathbb{E}(\tau_f) \cdot \sum_{n=0}^{\kappa_f'} {p_f}^n \cdot n$ is the expectation of energy consumption for failure execution under the condition that finally the correct result will be returned. $\mathbb{E}(e_f) \cdot \mathbb{E}(\tau_f) \cdot {p_f}^{\kappa_f'} \cdot \kappa_f'$ in right-hand side is the expected energy consumption of failure case. Since there is no correct result return, all the execution attempts are failed.

The expectation of failure penalty is

$$\mathbb{E}(C_f) = \gamma \cdot c_{pen} \cdot \mathbb{E}(\tau_f) \cdot \sum_{n=0}^{\kappa_f'} {p_f}^n \cdot n + (1-\gamma) \cdot (c_{pen} \cdot {p_f}^{\kappa_f'} \cdot \kappa_f' + c_{dt}). \quad (17)$$

The leftmost item is the failure penalty for the case that finally the correct result is returned, the item in the right-side is for the case that no correct result returns in the end of task lifespan.

The calculation of the $\mathbb{E}(\tau_f)$, $\mathbb{E}(e_f)$, $\mathbb{E}(p_f)$, e_s and τ_e for server-based cloudlet execution and ad hoc cloudlet execution are introduced separately as follows.

Server-based cloudlet execution. Given the uplink data transmission failure rate f_{up}, cloudlet execution failure rate f_{cl} and downlink execution failure rate f_{dw}, we first derive the probability that task execution fails in input data transmission phase, server-based cloudlet execution phase and output data transmission phase, respectively. The reduction result of probability that failure happens in input data transmission phase is calculated as

$$p_{up} = f_{up} \cdot (1 - f_{cloudlet}) \cdot (1 - f_{down}) / \Big(3 \cdot f_{up} \cdot f_{cloudlet} \cdot f_{down} - f_{up} + f_{cloudlet} + f_{down} - 2 \cdot \big(f_{up} \cdot f_{cloudlet} + f_{cloudlet} \cdot f_{down} + f_{up} \cdot f_{down}\big)\Big) \quad (18)$$

The denominator means that only at most one of the three types of failure happens at one time, because once the failure happens, the task processing procedure is interrupted and gives user a feedback to enter the failure recovery process. The numerator means that when the input data transmission failure happens, the execution failure and output data transmission failure won't happen anymore. In other words, the appearance of the three kinds of failure are mutual exclusive. The reduction results of execution failure rate $p_{cloudlet}$ and output data transmission failure rate p_{down} are calculated by the same way:

$$p_{cloudlet} = \big(1 - f_{up}\big) \cdot (1 - f_{down}) \cdot f_{ser} / \Big(3 \cdot f_{up} \cdot f_{ser} \cdot f_{down} + f_{up} + f_{ser} + f_{down} - 2 \cdot \big(f_{up} \cdot f_{ser} + f_{ser} \cdot f_{down} + f_{up} \cdot f_{down}\big)\Big). \quad (19)$$

$$p_{down} = \big(1 - f_{up}\big) \cdot (1 - f_{ser}) \cdot f_{down} / \Big(3 \cdot f_{up} \cdot f_{ser} \cdot f_{down} + f_{up} + f_{ser} + f_{down} - 2 \cdot \big(f_{up} \cdot f_{ser} + f_{ser} \cdot f_{down} + f_{up} \cdot f_{down}\big)\Big). \quad (20)$$

The expectation of failure rate is

$$\mathbb{E}(p_f) = p_{up} + p_{cloudlet} + p_{down}. \quad (21)$$

The expectation of failure energy consumption is

$$\mathbb{E}(e_f) = (p_{up} + p_{cloudlet} + p_{down}) \cdot e_{up} \cdot ip / BW_{up} \cdot \delta + p_{down} \cdot e_{down} \cdot op / BW_{down} \cdot \delta. \quad (22)$$

The expected time of failure execution is

$$\mathbb{E}(\tau_f) = (p_{up} + p_{cloudlet} + p_{down}) \cdot ip / BW_{up} \cdot \delta + p_{down} \cdot op / BW_{down} \cdot \delta. \quad (23)$$

Successful execution time is $\tau_s = ip / BW_{up} \cdot \delta + wl / s_{ser} + op / BW_{down} \cdot \delta$.

Ad hoc cloudlet execution. Ad hoc cloudlet offloading has two ways of data transmission: through RAN network or ad hoc network. Their immediate cost calculations are different. Here we introduce them separately. Given that the failure rate of a server node is f_{ad}. Supposed that there are v' server nodes which provide services, based on the backup technique, the code and data of the current task can be copied for v' folds. Therefore, the failure rate for task processing is ${f_{ad}}^{v'}$, the failure rate of data transmission from server nodes to access point is $f_{ad-ap} = {f_{up}}^{v'}$, the failure rate of data transmission from access point to server nodes is $f_{ap-ad} = {f_{down}}^{v'}$.

RAN network transmission. Failure can happen during the data transmission from user to the access point, from access point to the server node, from server node to the access point and from access point to the user. Since during the lifecycle of a task, the failure rate in each phase are independent of each other, we adopt the calculation method used in to derive the probability that failure happens in data transmission phase from user to access point p_{u-ap}, the data transmission phase from access point to server nodes p_{ap-ad}, the task execution phase p_{ad}, the data transmission phase from server nodes to access point p_{ad-ap} and the data transmission phase from access point to the user p_{ap-u}.

Therefore, the expected failure rate is

$$\mathbb{E}(p_f) = p_{u-ap} + p_{ap-ad} + p_{ad} + p_{ad-ap} + p_{ap-u} \tag{24}$$

Hence the expected RTT (Round-trip Time) for failure case is

$$\mathbb{E}(\tau_f) = (p_{u-ap} + p_{ap-ad} + p_{ad} + p_{ad-ap} + p_{ap-u}) \cdot \delta \cdot ip/BW_{up} + (p_{ap-ad} + p_{ad} + p_{ad-ap} + p_{ap-u}) \cdot \delta \cdot ip/BW_{down} + (p_{ad} + p_{ad-ap} + p_{ap-u}) \cdot wl/s + (p_{ad-ap} + p_{ap-u}) \cdot \delta \cdot op/BW_{down} + p_{ap-u} \cdot \delta \cdot op/BW_{down}. \tag{25}$$

where s is the processing speed. Since the energy consumption only comes from sending data back and forth, other phases will not generate extra energy consumption. The expected energy consumption for one failure case is derived as

$$\mathbb{E}(e_f) = (5 \cdot p_{u-ap} + 4 \cdot p_{ap-ad} + 3 \cdot p_{ad} + 2 \cdot p_{ad-ap} + p_{ap-u}) \cdot e_{up} \cdot \delta \cdot ip/BW_{up} + p_{ap-u} \cdot e_{down} \cdot \delta \cdot op/BW_{down}. \tag{26}$$

For RAN transmission, the scheduling time of one successful execution is $\tau_s = 2 \cdot \delta \cdot ip/BW_{up} + 2 \cdot \delta \cdot op/BW_{down} + wl/s$. The expected energy consumption is $e_s = e_{up} \cdot \delta \cdot ip/BW_{up} + \delta \cdot op/BW_{down}$.

Short-distance communication. The failure could happen in data transmission phase, task execution phase and data receiving phase. Similar to Eq. (19) and Eq. (20), we can derive the probability that failure happens in data sending phase p_s, task execution phase p_{ad} and data receiving phase p_r. Therefore, the expected failure rate is calculated as

$$\mathbb{E}(p_f) = p_s + p_{ad} + p_r. \tag{27}$$

Hence the expected RTT for failure case is

$$\mathbb{E}(\tau_f) = (p_s + p_{ad} + p_r) \cdot ip/r_{ad} + (p_{ad} + p_r) \cdot wl/s + p_r \cdot op/r_{ad}. \tag{28}$$

s is the maximum processing speed among the contact nodes. r_{ad} is the data rate of the short-distance communication. The expected energy consumption for one failure case is derived as

$$\mathbb{E}(e_f) = (3 \cdot p_s + 2 \cdot p_{ad} + p_r) \cdot e_{ad} \cdot ip/r_{ad} + p_r \cdot e_{ad} \cdot op/r_{ad}. \tag{29}$$

where e_{ad} is the unit energy consumption of short-distance communication.

Note that we calculating the expectation values above aims to get the expectation of total cost, hence,

$$\mathbb{E}(C(S,A)) = \omega_e \cdot \mathbb{E}(C_e) + \omega_r \cdot C_r + \omega_f \cdot \mathbb{E}(C_f). \tag{30}$$

where the value of ω_e, ω_r and ω_f usually set the same as Eq. (5). Similarly, we have $\omega_e + \omega_r + \omega_f = 1$.

4.4.1 Contact nodes pattern

We thus define a finite set $\{s_{ad}^1, s_{ad}^2, \dots, s_{ad}^l\}$ to describe the computation capabilities of all service node types in ad hoc network.

The number of contact nodes. The contact process of each node pair (i,j) is formulated as a Poisson process with an average contact rate of λ_{ij}. Supposed that in each decision period, the set of service nodes within user's communication range is represented as $\mathcal{V}$. What's more, each node

in $\mathcal{V}$ has a probability p_s to become a service node for user in the current decision period. For simplicity, we assume that the positions of nodes **will not** change too much during a decision period.

4.4.2 One-hop meeting delay estimation

When user decides to offload the current task to an ad hoc cloudlet, he chooses the service node which is the closest. The distance between user and service node is d. As user and service node can communicate with each other, $0 \le d \le R + r$. In this section, we estimate the one-hop meeting delay by considering user and the service nodes' moving patterns. Supposed that the velocity of user is $\overrightarrow{v_1}$, the velocity of the chosen service node is $\overrightarrow{v_2}$. The communication range of user is r, the communication range of service node is R. According to Manhattan mobility model, mobile users can only move either vertical and horizontal. Here we only need to consider the relative moving directions of two nodes.

1) User and service node moving over a 90-degree angle. Then the relative speed between user and service node is $\vec{v} = (\overrightarrow{v_1} - \overrightarrow{v_2})/|\overrightarrow{v_1} - \overrightarrow{v_2}| \cdot \sqrt{|\overrightarrow{v_1}|^2 + |\overrightarrow{v_2}|^2}$.
2) The horizontal movement of user and service node. Then the relative speed between user and service node is $\vec{v} = \overrightarrow{v_1} - \overrightarrow{v_2}$.

Hence, the maximum one-hop meeting delay estimation can be derived as

$$\tau_{max} = d/|\vec{v} + \vec{\varepsilon}| + (R + r)/(v + \varepsilon). \tag{30}$$

where $\vec{\varepsilon}$ is a vector with the same direction as $\vec{v}$ but its mode is small enough to be neglected.

As transmission through RAN network can be a complement of MANET, therefore, we adopt the following strategy: if $\tau_{max} \ge \mathbb{E}(\tau_{ad})$, then can use MANET transmission; otherwise, use RAN transmission.

5 Reinforcement learning based offloading framework

5.1 Computation offloading planning phase

The computation offloading problem can be seen as a multi-step reinforcement learning mission, a direct method is to view the action selection on each state as a K-rocker gambling machine problem, and use the accumulate reward in reinforcement learning instead of reward in K-rocker gambling machine. In other words, gambling machine algorithm can be used in each state: the attempt limit of each action and current accumulated reward need to be recorded. As the waiting time of tasks cannot be obtained in the offloading planning phase, we ignore this information for training but complement it in offloading running phase.

5.1.1 Modified ϵ- greedy algorithm

The basic ϵ- greedy algorithm is to strike a compromise using a probability ϵ. In each attempt, the agent explores with the probability ϵ, or utilizes the existing strategies with the probability $1 - \epsilon$. In both cases the rocker with the minimum reward is selected randomly or uniformly. In our modified version, for each state, the rocking is repeated for T times, and one available action with minimum reward value is selected.

Here we use $Q(S)$ to keep a record of the average reward of rocker S, where $S \in \mathbb{S}$. Supposed that rocker S has been tested for n times, the derived rewards are v_1, v_2,…, v_n. Then the average reward is $Q(S) = 1/n \cdot \sum_{i=1}^{n} v_i$. If we use this formula directly, we need to record n reward values. Each time we do the incremental calculation to $Q(S)$, $Q(S)$ is updated in each attempt for state S. Initially $Q_0(S) = 0$. For every $n \ge 1$, if the average reward after n-1 attempts is $Q_{n-1}(S)$, then after n^{th} attempt and get the reward value v_n, we have

$$Q_n(S) = Q_{n-1}(S) + 1/n \cdot \big(v_n - Q_{n-1}(S)\big). \tag{31}$$

In this paper, the maximum rocker number is $|\mathbb{S}| = |\mathbb{Z}| \cdot |\mathbb{V}| \cdot |\mathbb{Q}| \cdot |\mathbb{N}|$. The modified ϵ-greedy

algorithm is presented in figure 5.

Algorithm 1. Modified ϵ- greedy algorithm

Input: $C(S,A)$, attempts T, acquisition probability ϵ.

Output: map (S,A), accumulate reward r.

Begin

1: **Initialize** $r = 0$.
2: $\forall S \in \mathbb{S}$: $Q(S) = 0$, $count\ (S) = 0$
3: calculate the available action space for S: $A(S)$
4: **for** t = 1, 2, …, T **do**
5: **if** rand () < ϵ **then**
6: $k = rand(A(S))$ //*randomly choose by uniform distribution*
7: **else**
8: $k = argmin_A Q(S)$
9: **end if**
10: $v = \mathbb{E}(C\ (S,k))$
11: $r = r + v$
12: $Q(S) = \frac{Q(S)\times count(S)+v}{count(S)+1}$
13: $count\ (S) = count\ (S) + 1$
14: **end for**
15: **return** (S,A), r

End.

Figure 5. Modified ϵ- greedy algorithm.

r is the cumulative reward value and is initially set to 0 (line 1). The count value $count(S)$ and average reward are also set to 0 (line 2). For each state $S \in \mathbb{S}$, the attempts are repeated for T times. If the randomly generated value (in range $[0,1]$) is smaller than ϵ, k is any single available action (line 5-6), otherwise, the action with the minimum reward value is assigned to k (line 8). Since in offloading planning phase, task waiting time cannot be fetched from the environment, we use the expectation of reward value. The expectation of reward value $\mathbb{E}\big(C(S,k)\big)$ is calculated by Eq. (30). The calculated $Q(S)$ is the average reward value for state S (line 12). Finally, the offloading strategy (S,A) and cumulative reward value r are derived.

5.1.2 Modified soft-max algorithm

The basic soft-max algorithm is to strike a compromise between exploration and utility. If the reward of each rocker is of equal value, then the probability of being selected is almost the same for all rockers. The probability assignment is based on Boltzmann distribution:

$$P(S,A) = \frac{e^{\frac{Q(S)}{\rho}}}{\sum_{i=1}^{|A(S)|} e^{\frac{Q(S,i)}{\rho}}}. \tag{32}$$

where $A(S)$ is the set of available action of state S and $A \in A(S)$, $Q(S)$ is the average reward value for state S, $Q(S,i)$ is the average reward of current rocker i for state S, $\rho > 0$ is called "temperature", the smaller ρ means the higher probability that the rocker with higher reward value

will be chosen. The agent tends to "utilize only" when ρ goes to 0 and "explore only" when ρ goes to infinite. If the probabilities of some rockers are slightly higher than the others, then those rockers have higher probability to be chosen. The modified soft-max algorithm is presented in figure 6.

Algorithm 2. Modified soft-max algorithm

Input: $C(S,A)$, attempts count T, temperature parameter ρ.
Output: map (S,A), accumulate reward r.
Begin
1: **Initialize** $r = 0$.
2: $\forall S \in \mathbb{S}$: $Q(S) = 0$, $count\ (S) = 0$
3: calculate the available action space for S: $A(S)$
4: **for** t = 1, 2, …, T **do**
5: $k = rand(A(S), P(S,A))$
6: $v = \mathbb{E}(C(S,k))$
7: $r = r + v$
8: $Q(S) = \frac{Q(S) \times count(S) + v}{count(S)+1}$
9: $count\ (S) = count\ (S) + 1$
10: **end for**
11: **return** (S,A), r, $count\ (S)$
End.

Figure 6. Modified soft-max algorithm.

Similar to ϵ-greedy algorithm, firstly $Q(S)$ and $count(S)$ are initially are assigned to 0 (line 2) and the available action space for state S need to be derived (line 3). The only difference is the way to choose action (line 5) soft-max algorithm only randomly selects the action according to the probability calculated by Eq. (30).

5.2 Computation offloading running phase

In computation offloading running phase, the waiting time of tasks and fault tolerant strategy are considered to further revise the initial strategy derived from offloading planning phase. In offloading planning phase, the waiting time of task is hard to calculate because the timing relationship is ignored. However, in offloading running phase, the arrival task may not be executed immediately once it arrives, it has to wait until all former tasks finished their executions. Threshold based fault-tolerant strategy is also introduced in planning phase, it only influences the actual offloading strategy. The fault-tolerant offloading planning algorithm is introduced in figure 7.

Algorithm 3. Fault-tolerant Offloading Running Algorithm

Input: map (S,A), failure rate C_f, threshold value β, average reward $Q(S)$, $count(S)$.
Output: offloading strategy.
Begin
1: **while** queue is not empty **do**
2: **if** new tasks have arrived **then**

3: fill them into queue (if multiple tasks， then sort them according to lifespans)
4: the queue length increases
5: **end if**
6: $t_{sel} \leftarrow$ the current task in the queue head
7: select a state S according to the perception of the MEC system
8: select A according to the strategy of ϵ-greedy algorithm or soft-max algorithm
9: calculate C_f and $C(S,A)$ based on the actual execution
10: **if** $C_f > \beta$ **then**
11: choose action A' with minimum C_f value.
12: $A = A'$
13: **end if**
14: $rsa = C(S,A)$
15: use rsa to update $Q(S)$
16: delete t_{sel} from queue
17: **end while**
18: **return** offloading strategy
End.

Figure 7. Fault-tolerant Offloading Planning Algorithm.

Tasks arrive at the mobile device's queue at a Poisson process. When the queue is not empty, offloading planner fetches a task with the earliest arrival time (if there are multiple tasks arrive at the same time, randomly choose one task) to decide its execution site (line 6). Then the offloading planner monitors the system state S, including the zone user dwell in, the number of connections to the access point, the number of encountered mobile nodes and the length of queue (line 7). Then the action selection strategy is used. In line 11, the $C(S,A)$ and failure rate C_f are calculated. After finishing task execution, the actual $C(S,A)$ can be derived by Eq. (5) and can be used to update the average reward value $Q(S)$ (line 14-15). Finally, task is deleted from queue (line 16).

6 Performance evaluation

6.1 Parameter settings

To the best of our knowledge, there is no standard testbed that are suitable for evaluation. Therefore, we simulate the MEC environment and user behavior to test the proposed framework. The simulation is carried out in JAVA 8. In this section, we evaluate the performance of various algorithm through detailed evaluation. Table 2 shows the default values of main parameters used in all experiments. The confidence interval is set to 95% in all the simulations. In the sensitivity tests, we change the value of one of these parameters and set other parameters as the default values.

Table 2 Parameter Settings

Number of Zones	$Z = 20$
Queue Size	$Q = 10$
Task arrival rate	$\lambda_q = 0.25$
Contact node arrival rate	$\lambda_c = 15$
The connection rate to the access point	$\lambda_n = 100$

The Wi-Fi network capacity is normally distributed with means equal to 24 Mbps (IEEE 802.11g), the standard deviations equal to 10 Mbps. The unit energy consumption of uplink and downlink data transmission for Wi-Fi are 1.258 W and 1.181 W, respectively (IEEE 802.11g). Since the smart phone nowadays only equipped with Bluetooth for short-distance communication, the unit energy consumption of short-distance communication is 1 W (classical reference for Bluetooth). We consider the COTS (commercial off-the-shelf) of network device. For local execution in mobile device, N810 device is considered with a CPU frequency of 400×106 cycles/s, the unit energy consumption of task processing is 0.8W. We consider access points with dozens antennas; the total bandwidth of single access point is 300M. Other parameter setting are as follows: the failure rate of local execution is 0.05, the failure rate of server-based cloudlet execution 0.01, the failure rate of execution in ad hoc cloudlet is 0.1, the failure rate of data transmission 0.05. The simulation for each failure case is that calculating the cumulative probabilities for each possible situation and using a random value in range [0,1] to see that which situation is chosen to happen. For weight factor, we set $\omega_e = 0.3$, $\omega_r = 0.3$ and $\omega_f = 0.4$. since that the three optimization goals have different dimensions, we use data normalization method to turn the range of these three values to $[0, 1]$. In this paper, we consider general access points whose coverage range is 200m. The speed of user is 30m/s and the speed of service nodes are 40m/s. The range of short distant connection is 10m. We simulate urban environment where a mobile user moves through different zones at a speed. The probabilities of the four directions the user and service nodes may move on are 0.25. In our simulation, we utilize a zone-based RAN model where there are several predefined zones in the network. That is, nodes would stay in the neighborhood of a zone or move to the neighborhood of other zones with their own preferred probability. We use the method in [34] to simulate user's mobility.

The generation of AP map: The WiFi APs are randomly deployed spatially and each WiFi network covers at most four connected locations. We deploy 30 cloudlets across the whole map.

To stay close to the environment, we conduct a series of evaluations in three face detection applications whose parameters are listed in Table 3 which is also adopted in [16].

Table 3 Application profile for the evaluation

Workload	Average data size (byte)	Average android bytecode instructions (MI)	Numbers of tasks
S_L	725	5.8	500
S_H	650	24	500
B_H	5,000	29.5	500

6.2 Algorithms and settings in comparison

As is stated in introduction part, our concern is different from the existing proposed problems, there is no existing work appropriated for solving our problem directly. Hence, we implement two other baseline algorithms for comparison: random scheme and myopic scheme. We use $\epsilon-$greedy algorithm and soft-max algorithm to fill our framework separately.

- Random scheme (RND). The user randomly chooses the available execution site in each decision period.
- Myopic scheme (MYO). The mobile user makes a short-sighted decision only based on immediate

costs of the current decision period. Note that the chosen action should belong to the available action space in the current state. The decision is defined as follows:

$$A = argmin_A C(S, A). \quad (33)$$

- Which indicates that in each state, the action is taken when its according cost is the smallest.
- Offloading algorithm with $\epsilon-$greedy learning approach (OFG). Which means in both offloading planning phase and offloading running phase, the learning agent only use $\epsilon-$greedy learning approach to derive an optimal offloading strategy.
- Soft-max learning approach only (SO). Which means that only in offloading planning phase soft-max algorithm is adopted to learn an offloading strategy, and in offloading planning phase only use this offloading strategy to decide the execution site of each task.
- $\epsilon-$greedy learning approach only (GO). The same with SO, but use $\epsilon-$greedy learning approach to do the learning.

The two one-step methods cannot appear in the offloading framework at the same time (i.e. use greedy algorithm at offloading planning phase and use soft-max algorithm at offloading running phase), because their average reward values are calculated in different ways.

Furthermore, there are two alternative settings for the algorithms mentioned above. Since that the immediate cost are estimated in online algorithms, to test the validation of the estimation, we further consider the offline case where the immediate cost is the actual cost derived from task execution. However, in the everchanging MEC environment, reliability is a crucial consideration but may have negative affect on other optimized goals. Therefore, we set an option whether to adopt the fault tolerant mechanism or not.

Setting 1-1. Online. In offloading running phase, the immediate cost of each decision period is calculated according to section 4.3.

Setting 1-2. Offline. In offloading running phase, the immediate cost of decision period is calculated based on the environment simulated in the current decision period.

Setting 2-1. With fault tolerant. In offloading running phase, once the failure penalty of the selected action is larger than the given threshold, then choose the action with the minimum failure penalty.

Setting 2-2. Without fault tolerant. In decision period, choose an available action according to learning policy.

6.3 Evaluation results

To examine the efficiency of the proposed framework, we design multiple evaluations in order to answer the following questions: 1) is the expected cost model validated enough? 2) does the framework robust to different system states? and 3) the efficiency of fault tolerant strategy? Each data point is repeated 10 times for each application listed in table 3 to get the average value.

6.3.1 The impact of cost estimation deviation.

In order to prove that the online learning framework is validated in deriving an optimal strategy. In this experiment, we examine how the normalized average cost changes as the task number grows on all the three application tasks listed in table 3. The online algorithms are compared with offline algorithms to show the validation of estimated cost model and what we concern is how the performance of online algorithms improve in the process of task offloading (the control policy derived from online algorithm can update and adapt to the everchanging mobile context in theory). Therefore, offline algorithms are used as optimal ones and the evaluation studies how the performance of online algorithm to get close to the optimal one. For each application, the evaluation is repeated 10 times and finally get an average cost for all the three algorithms. The average cost is

examined for the case with fault tolerant mechanism and without fault tolerant mechanism. As is shown in figure 8, as the task number increases, the average cost of all the algorithm decrease, but the decrease rates of online algorithms are slightly bigger than offline algorithms. When the task number bigger than 400, the average cost of online algorithm and offline algorithms become very close. In terms of coverage speed, offline algorithm is faster than the online algorithm, they tend to be a stable value when the task number is 350, while the average cost of online algorithms becomes stable when the task number equals to 450. The reason behind could be that for offline algorithms, there is a deviation between estimated immediate cost and the actual immediate cost, it should take more time to revise the error brought by the estimation.

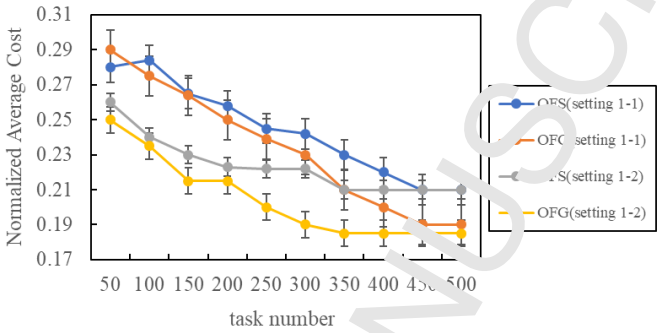


Figure 8. Normalized average cost versus task number (without fault tolerant).

Figure 9 is the case that considers the fault tolerant strategy, there are some shakes in the training process and shows more irregularly. This is because that the fault tolerant strategy doesn't influence the learning procedure and only affect the actual strategy for each real task. But in general, the average cost becomes closer when the task number increase. From this evaluation, it can be learned that as the task number grows, control policy derived by the online learning algorithm is continuously improved and finally close to the optimal one. In figure 10, it can be found out that after adopting fault tolerant strategy, the failure penalty becomes lower compared with the algorithms that do not use fault tolerant strategy. The gap between setting 2-1 and 2-2 can be reduced to a certain value because failure penalty is also one of the optimal goals in learning algorithms. However, the extra fault tolerant mechanism can further reduce the failure penalty with little harm to the general goal. Therefore, there is a trade-off between single goal optimization and unified goal optimization.

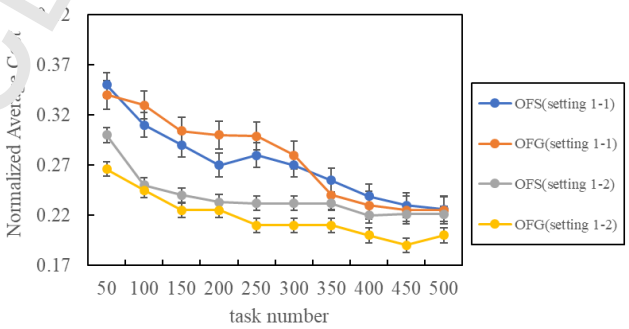


Figure 9. Normalized average cost versus task number (with fault tolerant).

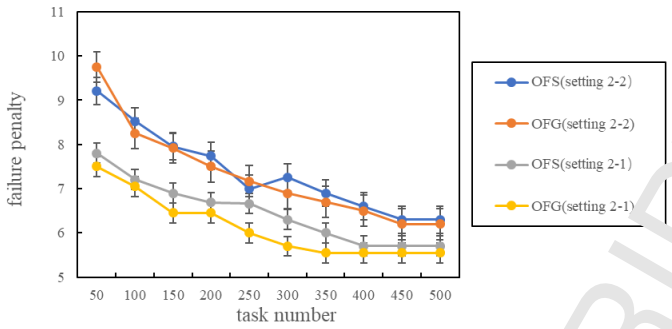


Figure 10. Failure penalty versus task number.

6.3.2 The validation of resource available.

Both server-based cloudlet and ad hoc cloudlet are considered in our work. In this evaluation, we investigate the performance of the proposed framework in different system states. We first evaluate the accuracy of the proposed online offloading approach. At each time slot, our method is used to expect possible reward values in making offloading decisions. To examine the efficiency of the expectation based online decision, we discuss the difference between online learning and offline learning.

Number of contact nodes in ad hoc cloudlets. In this evaluation, we change the system parameter and test the behavior of each algorithm in terms of offloading ratio and average cost. As can be seen in figure 11, as contact rate grows, the offloading ratios of all these four algorithms grow. The offline algorithms have lower average cost because of the inevitable deviation of the estimation. However, in offloading running phase we use the actual environmental feedbacks to revise the strategy, and it makes some improvements. For the learning algorithms, multi-steps learning approaches (OFS and OFG) performs slightly better than one-step learning approaches (SO and GO). However, the one-step learning approaches still perform better than RAD and MYO. But for the offloading ratio presented in figure 12, although remote execution (in ad hoc cloudlet or server-based cloudlet) means more monetary cost and risk, with more powerful computation capability, online algorithms are more likely to take "daring" schemes to allow more tasks to be executed remotely.

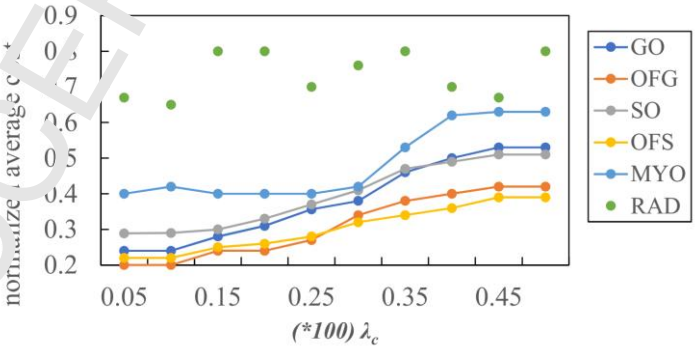


Figure 11. Normalized average cost versus contact rate λ_c.

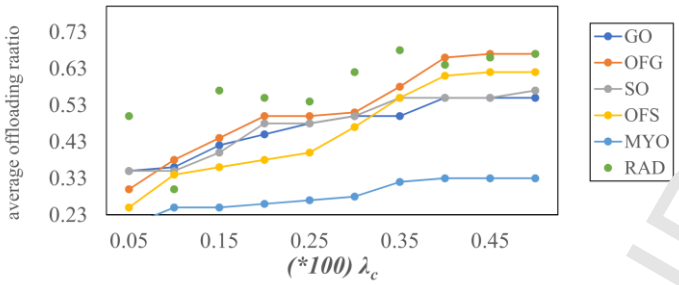


Figure 12. Average offloading ratio versus contact rate λ_c.

The congestion level of RAN. The evaluation result of server-based cloudlet availability is presented in figure 13 and figure 14. Higher λ_n means that the communication through RAN is more crowded. We evaluate the performance of all the online algorithms. For average cost, it can be seen that OFS and OFG always perform the best, GO and SO have lower average cost then MYO and RAD, RAD still shows no slight trend and the average cost is still high. Compared with OFS and OFG, SO and GO are less sensitive to the environmental changes for they have more gentle decline trends. Note that GO and SO have similar performance, but OFG have obviously lower average cost than OFS. This may due to the different learning strategies of greedy algorithm and soft-max algorithm. In offloading planning phase, they have fixed learning iterations to derive stable offloading strategy. However, in offloading running phase, the learning opportunity for each state are not equal due to the dynamic environment. Therefore, the offloading strategies could be different when adopting different learning strategies.

For offloading ratio, it can be seen that as the value of λ_n increase, the offloading ratio decrease, because the allocated bandwidth for each user becomes smaller when congestion level increase. At the beginning (when λ_n is smaller than 100) of each broken-line the average cost and offloading ratio are neither growing or falling, which is because that the user bandwidth lives up to the upper bound of allocated bandwidth. In most cases, the 95% confidence interval for the measured data is less than 16% of the corresponding mean values.

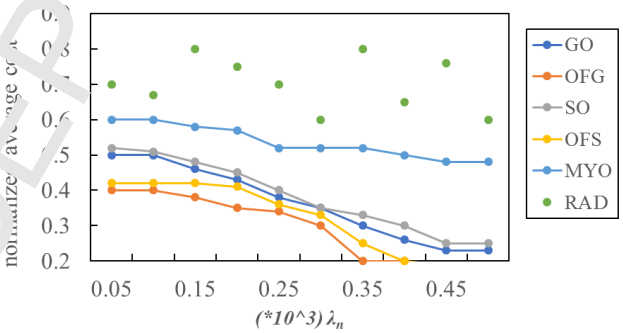


Figure 13. Normalized average cost versus connection rate λ_n.

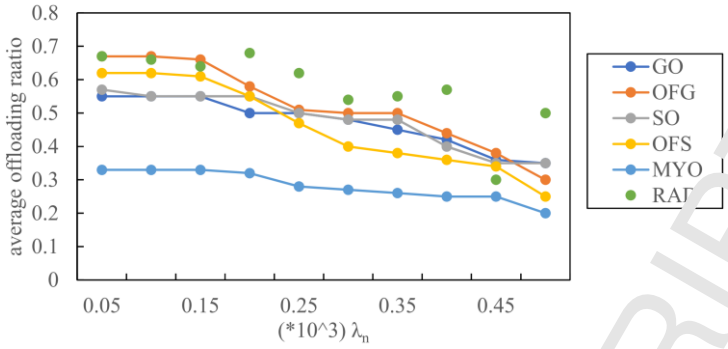


Figure 14. Average offloading ratio versus connection rate λ_n.

The impact of failure threshold β. This evaluation aims to proof the efficiency of fault tolerant strategy. In figure 15(a), it can be seen that when offloading threshold value increases, the average failure penalty decreases. However, when threshold value is bigger than 18, the failure penalty does not decrease any more. This phenomenon shows that the threshold-based fault tolerant strategy has the capability to reduce the failure penalty. However, the limitation could be caused by the task limited lifespan. As a real-time task has to wait in the queue until all its former tasks have finished their execution, the execution time is influenced by the former tasks. In other words, their failure rate is not independent of each other. Hence, our local optimal strategy can only derive limited fault tolerant strategy. But accompanied by the learning algorithm, the framework can derive a relatively lower penalty value. The threshold value corresponding to the 0.05 significance level. That is, we can accept the hypothesis at the 0.05 significance level that our proposed failure rate model fits with the statistical results from simulations.

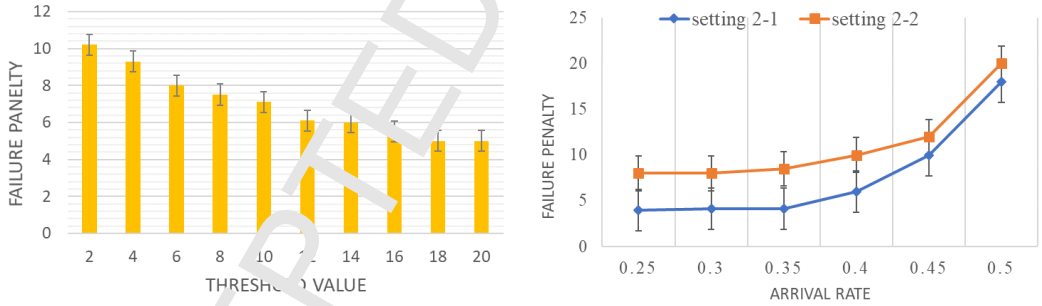


Figure 15. failure penalty versus threshold (a) value and failure penalty versus task arrival rate (b).

Based on the above analyze, we examine how the arrival rate of tasks influences the failure penalty in figure 15(b). Illustratively, the higher arrival rate means more tasks arrive in a certain time period, and the waiting time of each task will be longer. It can be seen that when arrival rate is smaller than 0.35, the failure penalty is quite stable and low; but the failure penalty grows sharply when the arrival larger than 0.45. this can be explained that when too many tasks arrive in a short period of time, the waiting time could be longer and lots of tasks are directly abandoned before execution. A lower failure rate can be achieved by increase the queue length. The fault tolerant strategy still plays a role for lowering failure rate. From this evaluation, it can be seen that the

threshold-based fault tolerant strategy along with failure recovery mechanism can reduce the failure penalty to some extends.

In summary, the above evaluations proof that 1) The proposed online learning method can derive a near optimal offloading control policy. The policy can be further improved and adapted to the mobile context in actual offloading process. 2) The online learning method shows robustness to the various mobile environment contexts, that is, when the congestion level of access point or the number of contact nodes varies, the algorithms based on the framework can always perform the best compared with other baseline algorithms. 3) The threshold-based fault tolerant strategy along with failure recovery mechanism can reduce the failure penalty to some extends.

7 Conclusions

Cloud of Things (CoT) is a significant paradigm for bridge cloud resource and mobile terminals. Mobile edge computing (MEC) is a supporting architecture for CoT and spread the benefits of computing resources to mobile devices' proximity. This paper presents an online computation offloading framework which consider the heterogeneous MEC system resource consists of server-based cloudlets and ad hoc cloudlets. We study the effect of network conditions on offloading decisions and find that the sparse deployment of ad hoc cloudlet and server-based cloudlet can reduce the offloading ratio, separately. The offloading framework includes the offloading planning phase and offloading running phase. In online running phase, the control policy tends to offload more tasks and can further reduce the average cost. We also investigate failure detection and recovery policies and use an expectation of cost model to derive online control policy. The numerical results show that the proposed online learning offloading method for mobile users can derive the improved optimal offloading scheme to reduce failure rate and the user's cost, which is to appreciate some real-time and location-aware applications such as video services which require a good user experience. It is particularly fit to the scenario where the user has to keep on moving for a long time until the policy can improve itself during the user's movement.

However, during the training process, it is hard to guarantee that the amounts of different types of samples are equal and this will affect the performance of offloading controller. For different users and different scenarios, it demands certain costs and needs users to move for a period of time. In our future work, techniques such as warm start of reinforcement learning or transfer learning can be adopted in single user's offloading scenario to speed up the learning process.

Acknowledgements

This work was supported by the National Key R&D Program of China (2016YFC0800803), the National Natural Science Foundation, China (No.61802167, 61872175, 61802095, 61572251, 61572162), the Fundamental Research Funds for the Central Universities. Jidong Ge is the corresponding author.

References

[1] M. Patel et al. (2014). Mobile-edge computing. In ETSI, France, White Paper.

[2] Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys and Tutorials, pages 2322-2358.

[3] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie. (2017). Mobile edge computing: A survey. IEEE Internet of Things, pages 450-465.

[4] Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K. B. (2017). Mobile edge computing: survey and research outlook.

[5] Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N. (2009). The case for vm-based cloudlets in mobile computing. IEEE Pervasive Computing, pages 14-23.

[6] Jin, H., Yan, S., Zhao, C., & Liang, D. (2016). Pmc 2 o: mobile cloudlet networking and performance analysis based on computation offloading. Ad Hoc Networks.

[7] Lyu, Xinchen, Tian, Hui, Jiang, Li, Vinel, Alexey, Maharjan, Sabita. (2017). Selective Offloading in Mobile Edge Computing for the Green Internet of Things. IEEE Network, pages 54-60.

[8] Li, T., Wang, K., Xu, K., Yang, K., Wang, H. (2017). On Efficient Offloading Control in Cloud Radio Access Network with Mobile Edge Computing. In Proc. *ICDCS*, pages 2258-2263.

[9] Xu J, Chen L, Ren S. Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing. (2017). IEEE Transactions on Cognitive Communications & Networking, pages 361-373.

[10] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal et al. (2014) Mobile-edge computing introductory technical white paper. White Paper, Mobile-edge Computing (MEC) industry initiative.

[11] Q. Xia, W. Liang, W. Xu. (2013). Throughput maximization for online request admissions in mobile cloudlet. in Proc. *LCN*, pages 589-596.

[12] Xu J, Chen L, Ren S. (2017). Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing. IEEE Transactions on Cognitive Communications and Networking, pages 361-373.

[13] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, pages 637–646.

[14] M. Chiang, T. Zhang. (2016). Fog and iot: An overview of research opportunities. IEEE Internet of Things Journal, pages 854–864.

[15] Yu-Jen Ku; Dian-Yu Lin; Chia-Fu Lee; Ping-Jung Hsieh; Hung-Yu Wei; Chun-Ting Chou; Ai-Chun Pang. (2017). 5G Radio Access Network Design with the Fog Paradigm: Confluence of Communications and Computing. IEEE Communication Magazine, pages 46-52.

[16] Chiang, M., Ha, S., Chih-Lin, I., Risso, F., Zhang, T. (2017). Clarifying fog computing and networking: 10 questions and answers. IEEE Communications Magazine, pages 18-20.

[17] Zhou, B., Dastjerdi, A. V., Calheiros, R., Srirama, S., Buyya, R. (2016). Mcloud: a context-aware offloading framework for heterogeneous mobile cloud. IEEE Transactions on Services Computing, pages 797-810.

[18] Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief. (2017). Mobile edge computing: Survey and research outlook. *arXiv preprint arXiv:1701.01090*.

[19] Zhang T. Data Offloading in Mobile Edge Computing: A Coalition and Pricing based Approach (2018). IEEE Access, pages 2760-2767.

[20] Tang L, He S. (2018). Multi-User Computation Offloading in Mobile Edge Computing: A Behavioral Perspective. IEEE Journal and Magazines, pages 48-53.

[21] Dinh, T. Q., Tang, J., La, Q. D., Quek, T. Q. S. (2017). Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Transactions on Communications, pages 3571-3584.

[22] Xu, J., Chen, L., Zhou, P. (2018). Joint service caching and task offloading for mobile edge computing in dense networks.

[23] Wang, C., Liang, C., Yu, F. R., Chen, Q., Tang, L. (2017). Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. IEEE Transactions on Wireless Communications, pages 4924-4938.

[24] Tao, X., Ota, K., Dong, M., Qi, H., Li, K. (2017). Performance guaranteed computation offloading for mobile-edge cloud computing. IEEE Wireless Communications Letters, pages 774-777.

[25] Guo, H., Liu, J., Qin, H., Guo, H., Liu, J., Qin, H., et al. (2017). Collaborative Computation Offloading for Mobile-Edge Computing over Fiber-Wireless Networks. In Proc. *GLOBECOM*.

[26] Wang C, Li Y, Jin D. (2014). Mobility-Assisted Opportunistic Computation Offloading. IEEE Communications Letters, pages1779-1782.

[27] Chen S, Wang Y, Pedram M. (2014). Optimal offloading control for a mobile device based on a realistic battery model and semi-Markov decision process. Pages 369-275.

[28] Zhang Y, Niyato D, Wang P. (2015). Offloading in Mobile Cloudlet Systems with Intermittent Connectivity. IEEE Transactions on Mobile Computing, pages 2516-2529.

[29] Terefe, M. B., Lee, H., Heo, N., Fox, G. C., Oh, S. (2016). Energy-efficient multisite offloading policy using markov decision process for mobile cloud computing. Pervasive and Mobile Computing, pages 75-89.

[30] Sun, X., Ansari, N. (2017). Latency aware workload offloading in the cloudlet network. IEEE Communications Letters, pages 1481-1484.

[31] Hyytia, E., Spyropoulos, T., Ott, J. (2015). Offload (only) the right jobs: Robust offloading using the Markov decision processes. IEEE World of Wireless, Mobile and Multimedia Networks, pages 1-9.

[32] Liu, D., Khoukhi, L., Hafid, A. S. (2017). Data Offloading in Mobile Cloud Computing: A Markov Decision Process Approach. In Proc. *ICC*, pages 1-6.

[33] Hu, Z., Lu, Z., Li, Z., Wen, X. (2016). Adaptive network selection based on attractor selection in data offloading. In Proc. *WCNC*, pages 1-6.

[34] Yuan Q, Cardei I, Wu J. (2009). Predict and relay: an efficient routing in disruption-tolerant networks. In Proc. *MOBIHOC*, pages 95-104.

[35] D. Moltchanov. (2012). Distance distributions in random networks. Ad Hoc Network, pages 1146–1166.

[36] F. Baccelli, B. Blaszczyszyn, P. Muhlethaler. (2006). An Aloha protocol for multihop mobile wireless networks. IEEE Transactions on Information Theory, pages 421–436.

[37] Suraj, R., Tapaswi, S., Yousef, S., Pattanaik, K. K., & Cole, M. (2016). Mobility prediction in mobile ad hoc networks using a lightweight genetic algorithm. Wireless Networks, 22(6), 1797-1806.

[38] Gao, W., & Cao, G. (2010). On Exploiting Transient Social Contact Patterns for Data Forwarding in Delay-Tolerant Networks. IEEE International Conference on Network Protocols, ICNP 2010, Kyoto, Japan, 5-8 October (Vol.12, pp.193-202). DBLP.

Highlights:

The major revised contents are summarized as following：

1. According to the reviewer's comment, we reconstruct the abstract content and answer the specific problems to be solved: "describe and evaluate a solution to handle the computation offloading problem during user mobility and minimize offloading failure rate in heterogeneous network".

2. Introduction part is rewritten. Firstly, the concepts of CoT and MEC are introduced, and we further explain why MEC can be connected to CoT: "it enables cloud computing service at the edge of cellular network, which provides timely and on-demand access for mobile applications during users' moving". Then computation offloading in MEC environment is introduced. We use a sketch graph to explain the scenarios that our approach fitted to.

3. Problems about English expression and grammar have been carefully checked. The expression problems the reviewers mentioned have been carefully revised and we further correct several expression problems after checking throughout the paper.

4. The construct of the model part are revised. Section 3.2 and Section 3,3 is merged and renamed as "Mobile edge system". Correspondingly, we complement the introduction to Mobile edge system and explicit its border.

5. the feasibility of the proposed problem and methods are further explained by presenting some examples, marking where the important parameter settings come from and claiming the practical significance of simulation result.

More specific descriptions of revised parts are illustrated and explained in concrete responses to reviewers' comments. Below we have italicized the reviewers' comments in order to provide the context for our responses and we show our responses in blue. Thanks again from the bottom of our heart for your hearty decision on our research paper.

Feifei Zhang is a PhD student at Software Institute, Nanjing University, under the supervision of Prof. Jidong Ge and Prof. Bin Luo. Her research interests include cloud computing, workflow scheduling and edge computing.

Jidong Ge is an Associate Professor at Software Institute, Nanjing University. He received his PhD degree in Computer Science from Nanjing University in 2007. His current research interests include cloud computing, workflow scheduling, software engineering, workflow modeling, process mining. His research results have been published in more than 80 papers in international journals and conference proceedings including IEEE TSC, JASE, FGCS, JSS, Inf. Sci., JNCA, JSEP, ESA, ICSE, GLOBECOM, IWQoS, APSEC, ICSSP, HPCC, SEKE, PRICAI etc.

Chifong Wong is an undergraduate student at Software Institute, Nanjing University, China since 2014. His main subject is operating system, embedded system and software architecture. His current research interests include system software, compiling technology, machine learning and Artificial Intelligent.

Chuanyi Li is an Assistant Professor at Software Institute, Nanjing University, China. He received his PhD degree and B. S. degree in Software Engineering from Nanjing University, China in 2017 and in 2012 respectively. His research interests include software engineering, systems and software quality assurance, process modeling, simulation and improvement, process mining. His research results have been published in IEEE TSC, FGCS, JSS, Inf. Sci. etc.

Xingguo Chen received Ph.D. degree in computer application technology from the Department of Computer Science and Technology, Nanjing University, China, in 2013. He is currently a lecturer and master instructor at the School of Computer Science and Technology, School of Software, Nanjing University of Posts and Telecommunications. His research interests include game AI and reinforcement learning.

Bin Luo is a full professor at Software Institute, Nanjing University. His main research interests include cloud computing, computer network, workflow scheduling, software engineering. His research results have been published in more than 80 papers in international journals and conference proceedings including IEEE TSC, ACM TOIST, FGCS, JSS, Inf. Sci., ESA etc. He is leading the institute of applied software engineering at Nanjing University.

Victor Chang is an Associate Professor in Information Management and Information Systems at International Business School Suzhou (IBSS), Xi'an Jiaotong Liverpool University, China. He is a Director of Ph.D. Program and the 2016 European and Cloud Identity winner of "Best Project in Research". Victor Chang was a Senior Lecturer in the School of Computing, Creative Technologies at Leeds Beckett University, UK and a visiting Researcher at the University of Southampton, UK. He is an expert on Cloud Computing and Big Data in both academia and industry with extensive experience in related areas since 1998. He completed a PGCert (Higher Education) and Ph.D. (Computer Science) within four years while working full-time. He has over 100 peer-

reviewed published papers. He won £20,000 funding in 2001 and £81,000 funding in 2009. He was involved in part of the £6.5 million project in 2004, part of the £5.6 million project in 2006 and part of a £300,000 project in 2013. He won a 2011 European Identity Award in Cloud Migration and 2016 award. He was selected to present his research in the House of Commons in 2011 and won the best papers in 2012 and 2015. He has demonstrated ten different services in Cloud Computing and Big Data services in both of his practitioner and academic experience. His proposed frameworks have been adopted by several organizations. He is the founding chair of international workshops in Emerging Software as a Service and Analytics and Enterprise Security. He is a joint Editor-in-Chief (EIC) in International Journal of Organizational and Collective Intelligence and a founding EIC in Open Journal of Big Data. He is the Editor of a highly prestigious journal, Future Generation Computer Systems (FGCS). His security paper is the most popular paper in IEEE Transactions in Services Computing and his FGCS paper has one of the fastest citation rate. He is a reviewer of numerous well-known journals and has published three books on Cloud Computing which are available on Amazon website. He is a keynote speaker for CLOSER 2015/WEBIST2015/ICT for Ageing Well 2015 and has received positive support. He is the founding chair of IoTBD 2016 www.iotbd.org and COMPLEXIS 2016 www.complexis.org conferences.